树形dp

2019年4月17日 15:06

-
- 树形dp

一. 树形dp

1. 以节点从申到浅，子树从小到大作为dp阶段
2. 第一维通常是节点编号，代表了该节点为根的子树
3. 一般是递归地求解子结点，在往根回溯，转移状态
4. 即使是无向无根树也可通过自定义根来转换成树

```
int n,m,tot=1;      //点数，边数，边号
int fst[MN],to[MN<<1],nxt[MN<<1],val[MN<<1];
inline void add(int x,int y,int z=0){
     to[++tot] =y, nxt[tot] =fst[x], fst[x] =tot;
     //val[tot] =z;
}
void dp(int x,int f){
     for__(x,2,n){
          for(int i=fst[x]; i; i=nxt[i]){
               int y=to[i];
               if(!d[y]){

               }
          }
     }
}
```

二. 例题

1. maxmin替罪羊：第一行10表示各结点是取子树的最大值还是最小值，第二行表示除了根结点的各结点的父结点编号。设叶结点有k个，分别占有正整数1~k，求根节点能获得的最大数
   i. 得利于父结点编号一定小于子结点，逆序遍历各结点就能保证一定是更新完了叶子才转移给父结点
   ii. d[i]存储i可能拿到编号第i大的数字
   iii. 对于max结点，一定能拿到子结点中的最小值，此时，让未被转移的子结点实义上拿到的d[i]都大于被转移的d[i]即可
   iv. 对于min结点，就只能拿子结点的最大值，且此时未被转移的子结点实义上拿到的d[i]都小于d[i]，所以被转移的d[i]必须同时大于其他d[i]且要保证没有重复，实际上真正转移给父结点的应是所有子结点的和
   v.
```
for__(i,1,n)
        scanf("%d",M+i);
for__(i,2,n){
        scanf("%d",&f);
        nxt[i]=fst[f];
        fst[f]=i;}
int k=0;
rof__(i,n,1){
        if(fst[i]==0)       //子结点
```

```
                d[i]=1,             //假设有机会放第1大的数
                ++k;
            else if(M[i]){      //能拿Max of子结点
                d[i]= 1<<30;
                for(int t= fst[i]; t; t=nxt[t])
                    d[i]=min(d[i], d[t]);}
            else      //拿min of子结点，此时子结点假设之和为最小假设
                for(int t= fst[i]; t; t=nxt[t])
                    d[i]+= d[t];}
        cout<<k- d[1]+ 1;
```

-
- 二次扫描与换根

一. 两次搜索换根法：

i. 第一次先设1为根，dfs求出单源最短路和各子树大小，从单源最短路又可求权

ii. 然后dfs换根：设新根为y，y子树上的点可直接到根；y子树外的点需先到原根x，再从原根x到新根y。sz[y]个结点到根的距离少了d[x][y]=val[i]；n-sz[y]个结点到根的距离多了d[x][y]=val[i]，总变化=(n- 2* sz[y])* val[i]

iii. 注意出现乘号的地方可能溢出

iv.
```
int n;
int nxt[MN],fst[MN],tmn[MN],val[MN];
//边结点指针域，链表头，边终点，边权
ll d1[MN];              //1为根的单源最短路
int sz[MN];             //各结点为根的子树上的结点数量
ll ans[MN];             //各结点为根时的路总和
int ansidx= 1;    //路总和的最小值的结点序号

void dfs(int x){  //遍历x的子树，求单源最短路和子树大小
    ++sz[x]; //结点本身也算上
    for(int i= fst[x]; i; i= nxt[i]){
        int y= tmn[i];
        d1[y]= d1[x]+ val[i];
        dfs(y);
        sz[x]+= sz[y];}}
void dfs2(int x){ //换x的子结点y为根，求总路程和
    for(int i= fst[x]; i; i= nxt[i]){       //开始换根
        int y= tmn[i];
        ans[y]= ans[x]+ (ll)(n- 2* sz[y])* val[i];
        dfs2(y);
        if(ans[ansidx] > ans[y])
            ansidx= y;
        else if(ans[ansidx] == ans[y])
            ansidx= min(ansidx, y);}}
```

v.
```
cin>>n;
int x,y,v;
for_(i,1,n){
    cin>>x>>y>>v;
    if(x>y)
        swap(x,y);      //保证y>x
```

```
        tmn[i]= y;
        val[i]= v;
        nxt[i]= fst[x];
        fst[x]= i;}
    dfs(1);             //先以1为根，更新d1和sz
    for__(i,2,n)
        ans[1] += d1[i];
    dfs2(1); //再换根
    cout<< ansidx<< endl<< 2*ans[ansidx];
```

二. 例题：结点带权树最大源

i.
```
int n,m,t;
ll e[MN],d1[MN],sz[MN];
ll ans[MN];
bool vis[MN];
vector<int> tmn[MN],val[MN];
string s;
void dfs1(int x){
    sz[x]= e[x];
    vis[x]= 1;
    for_(i,0,tmn[x].size()){
        int y= tmn[x][i];
        if(!vis[y])
            d1[y]= d1[x]+ val[x][i],
            dfs1(y),
            sz[x]+= sz[y];}}
void dfs2(int x){
    vis[x]= 1;
    for_(i,0,tmn[x].size()){
        int y= tmn[x][i];
        if(!vis[y]){
            ans[y]= ans[x]+ (sz[1]-2*sz[y])* val[x][i];
            dfs2(y);}}}
```

ii.
```
cin>>n;
for__(i,1,n)
    cin>>e[i];
int x,y,v;
for_(i,1,n)
    cin>>x>>y>>v,
    tmn[x].push_back(y),
    val[x].push_back(v),
    tmn[y].push_back(x),
    val[y].push_back(v);
dfs1(1);
for__(i,2,n)
    ans[1]+= d1[i]*e[i];
ms(vis,0);
dfs2(1);
int ret1= 1;
for__(i,2,n)
    if(ans[i]<ans[ret1])
        ret1= i;
cout<<ret1<<endl<<2*ans[ret1];
```

三.